Fog Effect, Mouse picking
and
GLSL Intrinsic Functions

MEDIA
DESIGN
SCHOOL
GAME
DEV

Fog

- To simulate certain types of weather conditions in our games, we need to be able to implement a fog effect.
- Provides some fringe benefits
- *Popping* refers to an object that was previously behind the far plane all of a sudden coming in front of the frustum, due to camera movement, and thus becoming visible

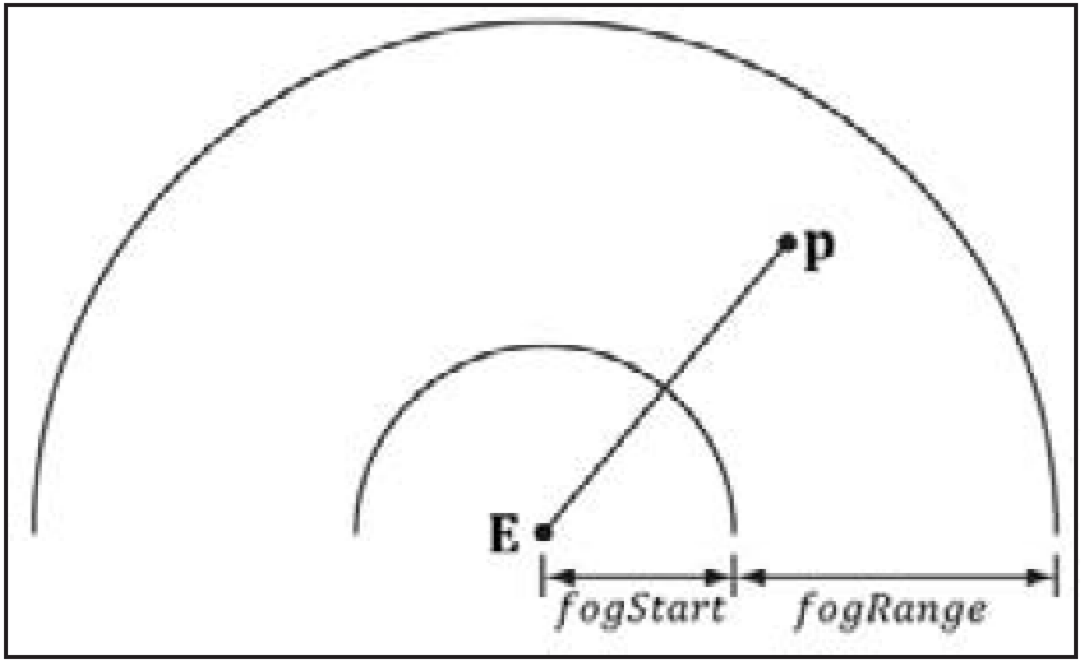
Fog

- if your scene takes place on a clear day, you may still wish to include a subtle amount of fog at far distances, because, even on clear days, distant objects such as mountains appear hazy and lose contrast as a function of depth
- We can use fog to simulate this *atmospheric perspective* phenomenon.

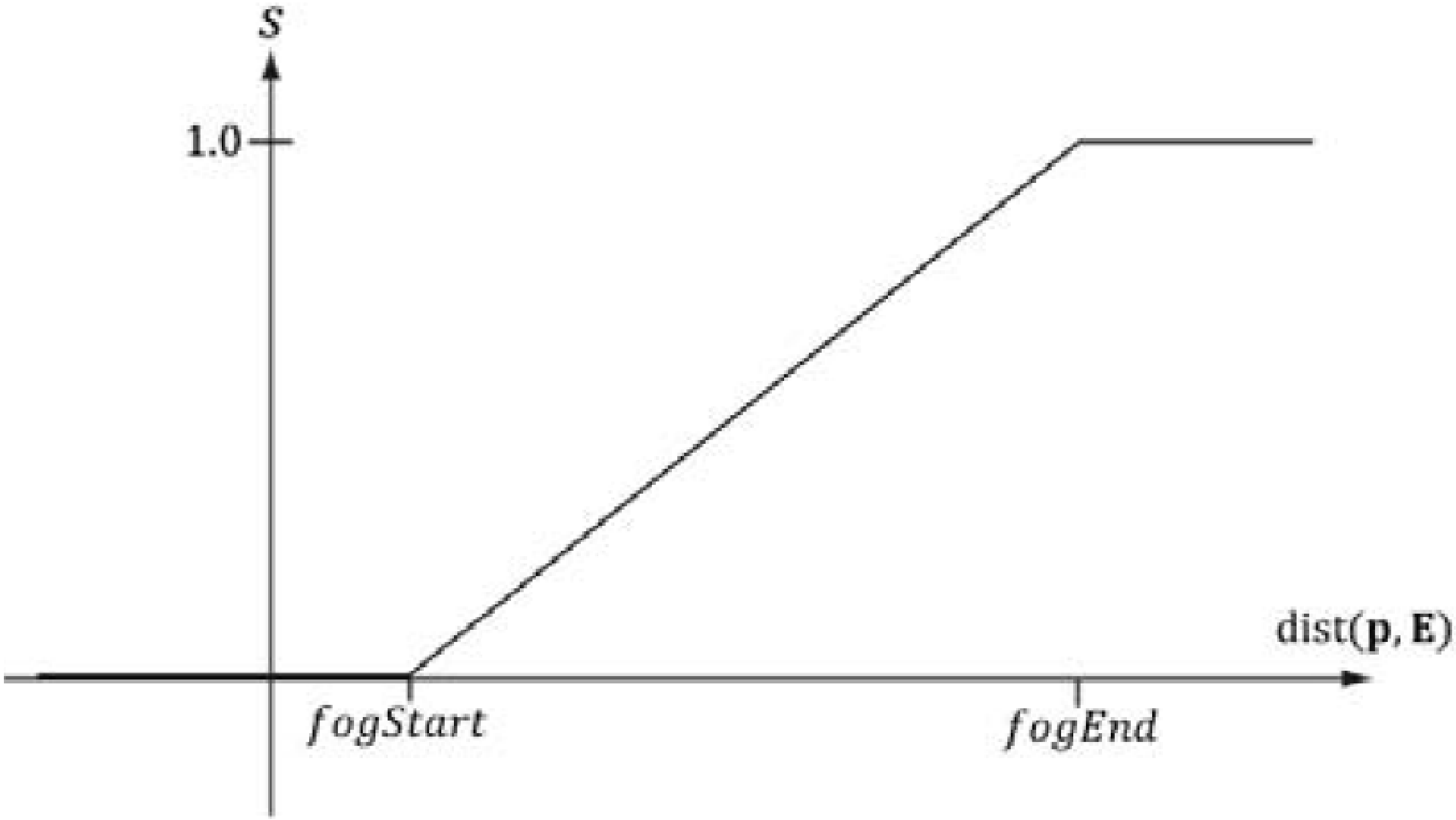
Fog

- We specify
  - a fog color,
  - a fog start distance from the camera,
  - and a fog range
- Then the color of a point on a triangle is a weighted average of its usual color and the fog color

Fog



Fog



Fog

$$\frac{dist(\mathbf{p},\mathbf{E}) - fogStart}{fogRange}$$

Fog

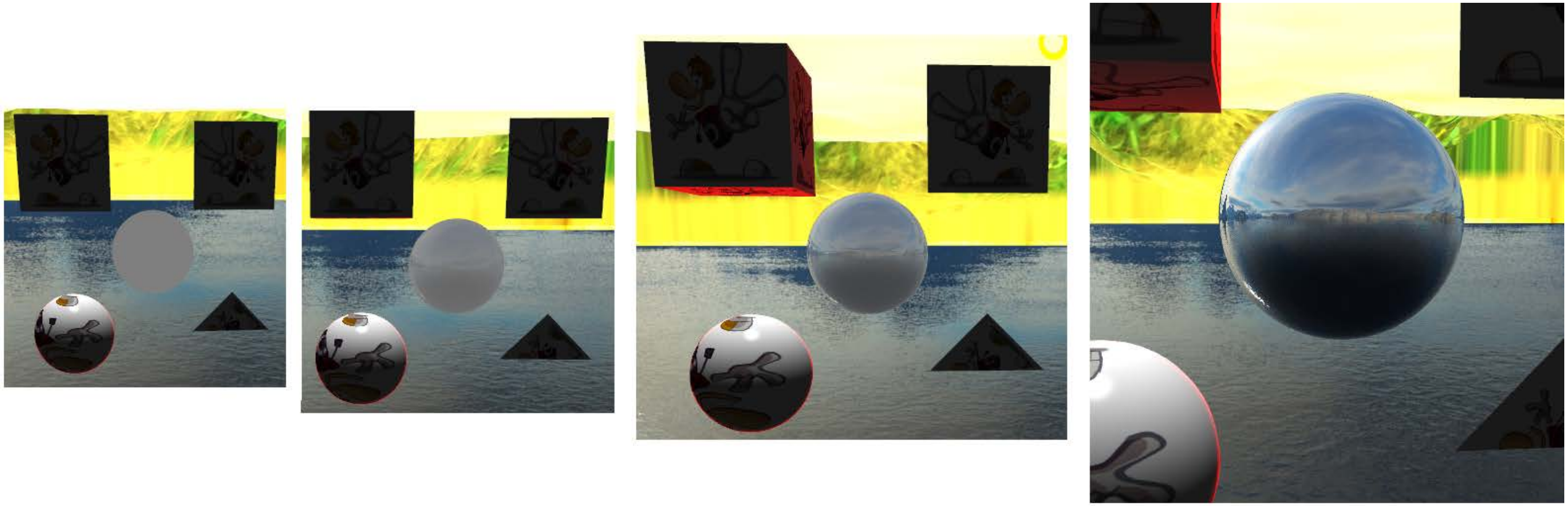
```
//** Vertex Shader
vec4 mWorldPos = model *vec4(position, 1.0);
gl_Position =  proj * view * worldPos;

//** fragment shader
float d = distance(mWorldPos.xyz, cameraPos);
float lerp = (d - 5.0f)/10.f;
lerp = clamp(lerp, 0.0, 1.0);

vec4 vFogColor = vec4(0.5f, 0.5f, 0.5f, 1.0f);

color = mix(color, vFogColor, lerp);
```

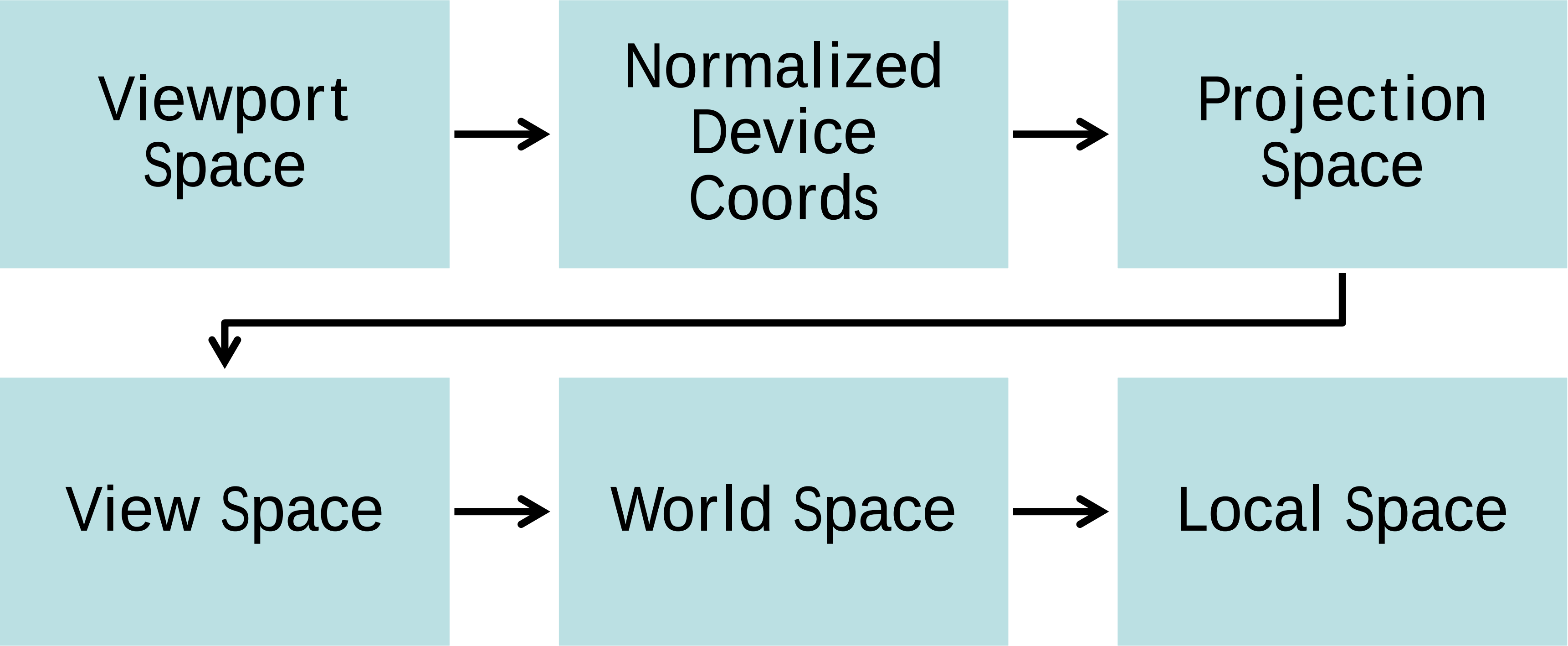
Fog



3D Mouse Picking

MEDIA DESIGN SCHOOL GAME DEV

Mouse Picking



Viewport Space

Normalized Device Coords

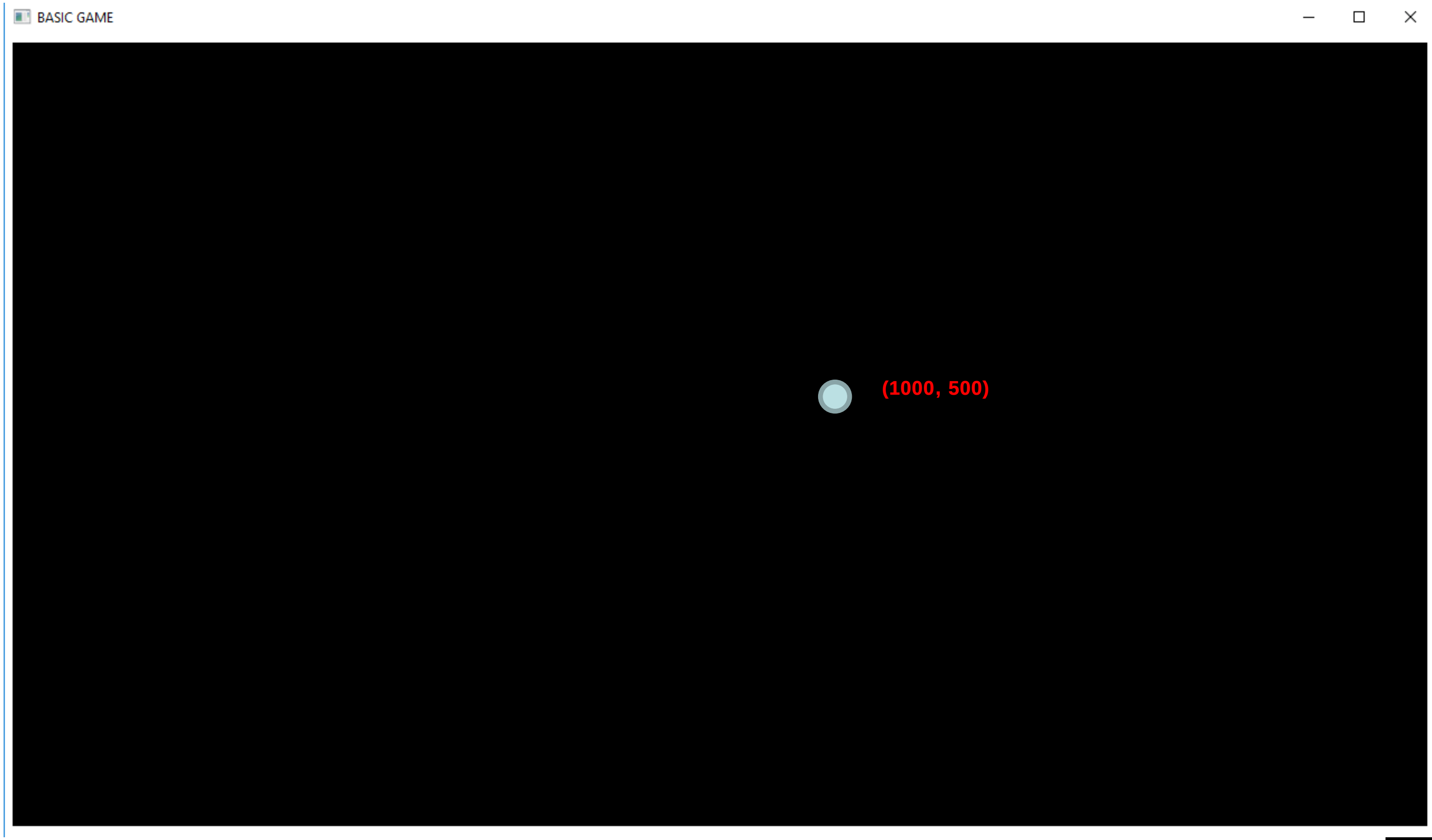
Projection Space

View Space

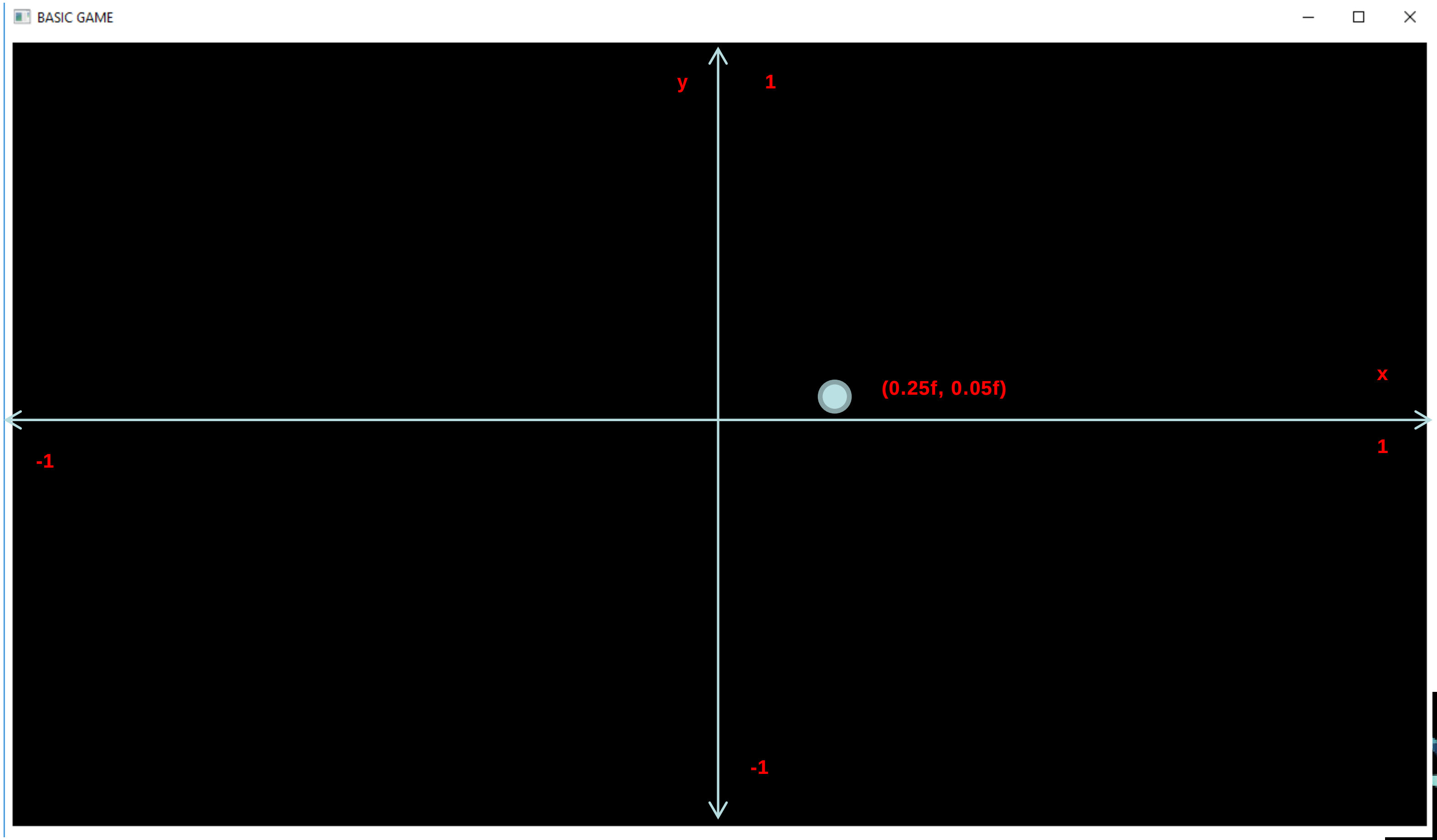
World Space

Local Space

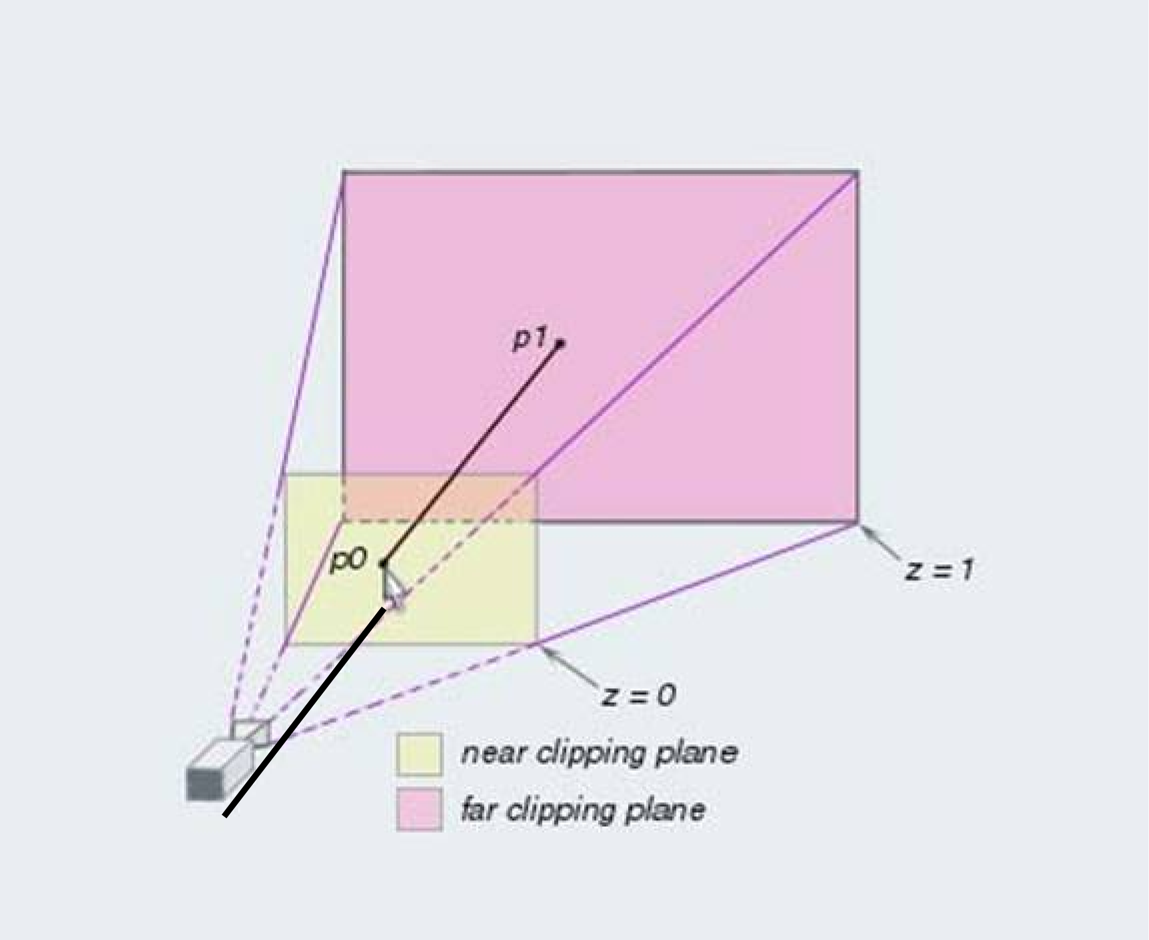
Viewport Space



Normalized Device Coords



Projected Ray



Mouse Picking

- Create variables in main.cpp to store values

  ```
  glm::vec3 rayDirection;
  float mouseY;
  float mouseX;
  ```

- In mousePassive function set the values of mouseX and mouseY. Converted to NDC.

```
mouseX = (2.0f * x) / (float)Utils::WIDTH - 1.0f;
mouseY = 1.0f - (2.0f * y) / (float)Utils::HEIGHT;
```

Mouse Picking

- Create new function **updateMousePicking** add following to it.

```
bool updateMousePicking(){
//screen pos
glm::vec2 normalizedScreenPos = glm::vec2(mouseX, mouseY);

//screenpos to Proj Space
glm::vec4 clipCoords = glm::vec4(normalizedScreenPos.x, normalizedScreenPos.y, -1.0f, 1.0f);

//Proj Space to eye space
glm::mat4 invProjMat = glm::inverse(camera->getprojectionMatrix());
glm::vec4 eyeCoords = invProjMat * clipCoords;
eyeCoords = glm::vec4(eyeCoords.x, eyeCoords.y, -1.0f, 0.0f)
```

Mouse Picking

```
    //eyespace to world space
    glm::mat4 invViewMat = glm::inverse(camera->getViewMatrix());
    glm::vec4 rayWorld = invViewMat * eyeCoords;
    rayDirection = glm::normalize(glm::vec3(rayWorld));

    //add code to check
    // intersection with other objects
}
```

- Following code checks intersection of ray with a sphere of radius 1.0f.
- Similarly intersection with other shapes can be added.
- Most Physics engines has code for checking intersection with physics objects.

Check intersection with Object

```
    float radius = 1.0f;
    glm::vec3 v = sphere->getPosition() - camera->getCameraPosition();

    float a = glm::dot(rayDirection, rayDirection);
    float b = 2 * glm::dot(v, rayDirection);
    float c = glm::dot(v, v) - radius * radius;
    float d = b * b - 4 * a* c;

if (d > 0) {
    float x1 = (-b - sqrt(d)) / 2;
    float x2 = (-b + sqrt(d)) / 2;
    if (x1 >= 0 && x2 >= 0) return true; // intersects
    if (x1 < 0 && x2 >= 0) return true; // intersects
}else if (d <= 0) {
    return false;// no intersection
}
```

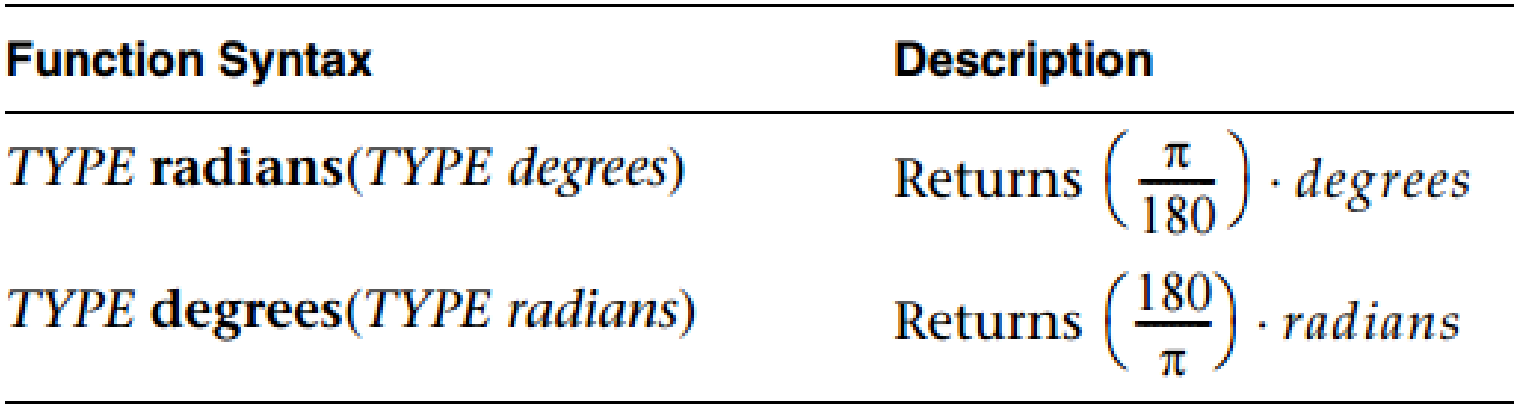
Mouse Picking

- Add updateMousePicking function to your update function.

Built-In OpenGL Shading Language Functions

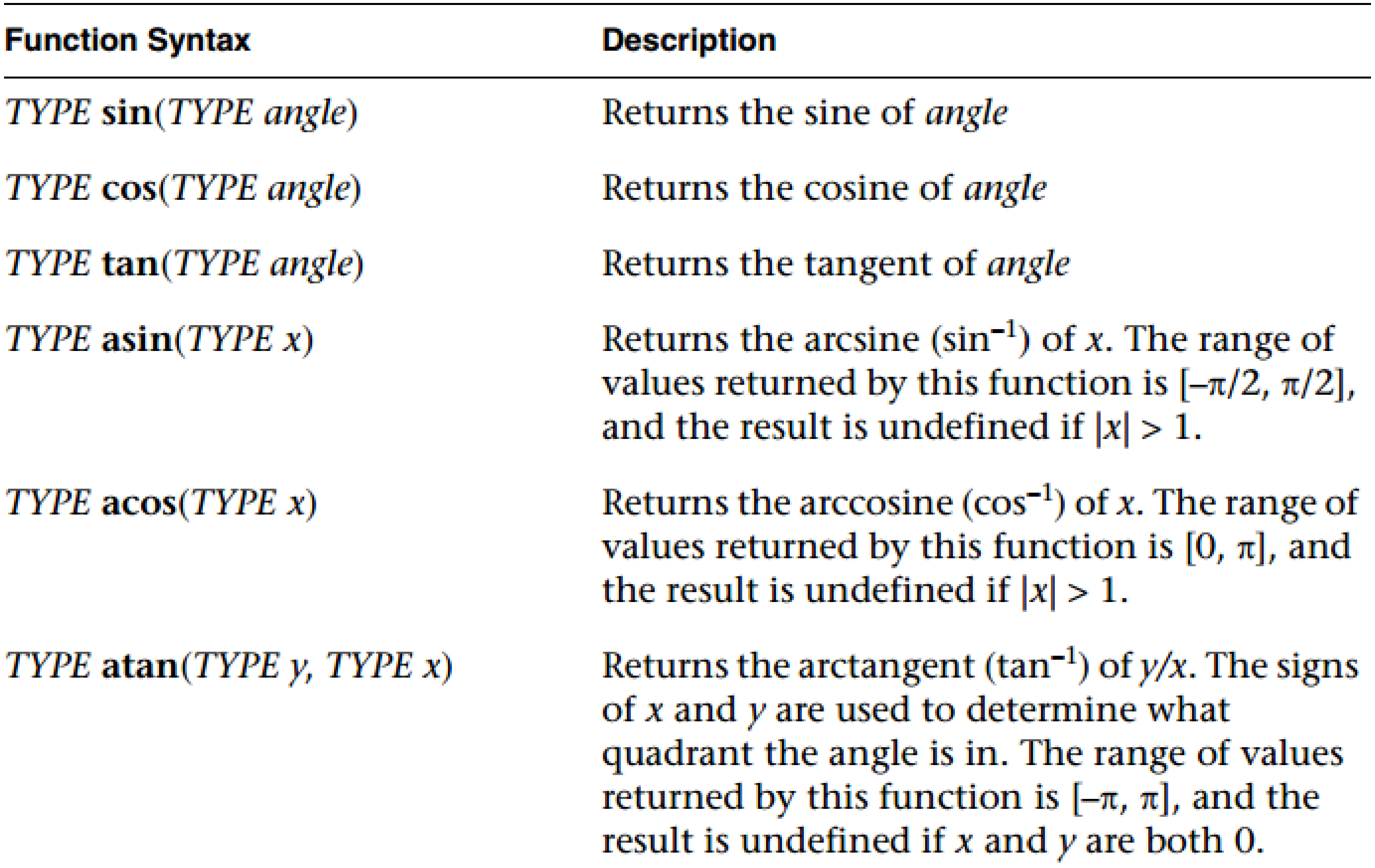
Built-In OpenGL Shading Language Functions

- Angle Conversion and Trigonometry Functions



Function Syntax	Description
TYPE **radians**(*TYPE degrees*)	Returns $\left(\frac{\pi}{180}\right) \cdot degrees$
TYPE **degrees**(*TYPE radians*)	Returns $\left(\frac{180}{\pi}\right) \cdot radians$

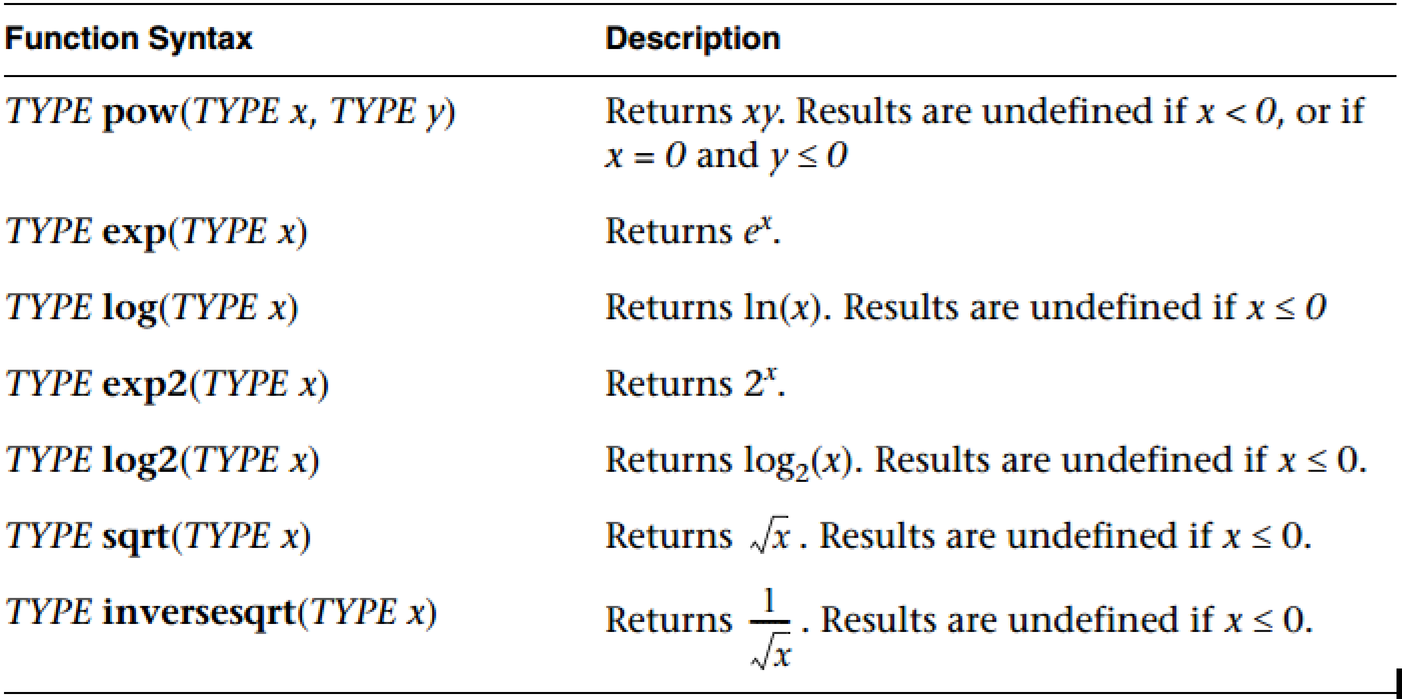
Built-In OpenGL Shading Language Functions



Function Syntax	**Description**
TYPE **sin**(*TYPE angle*)	Returns the sine of *angle*
TYPE **cos**(*TYPE angle*)	Returns the cosine of *angle*
TYPE **tan**(*TYPE angle*)	Returns the tangent of *angle*
TYPE **asin**(*TYPE x*)	Returns the arcsine ($\sin^{-1}$) of x. The range of values returned by this function is $[-\pi/2, \pi/2]$, and the result is undefined if $\lvert x \rvert > 1$.
TYPE **acos**(*TYPE x*)	Returns the arccosine ($\cos^{-1}$) of x. The range of values returned by this function is $[0, \pi]$, and the result is undefined if $\lvert x \rvert > 1$.
TYPE **atan**(*TYPE y*, *TYPE x*)	Returns the arctangent ($\tan^{-1}$) of y/x. The signs of x and y are used to determine what quadrant the angle is in. The range of values returned by this function is $[-\pi, \pi]$, and the result is undefined if x and y are both 0.

Built-In OpenGL Shading Language Functions

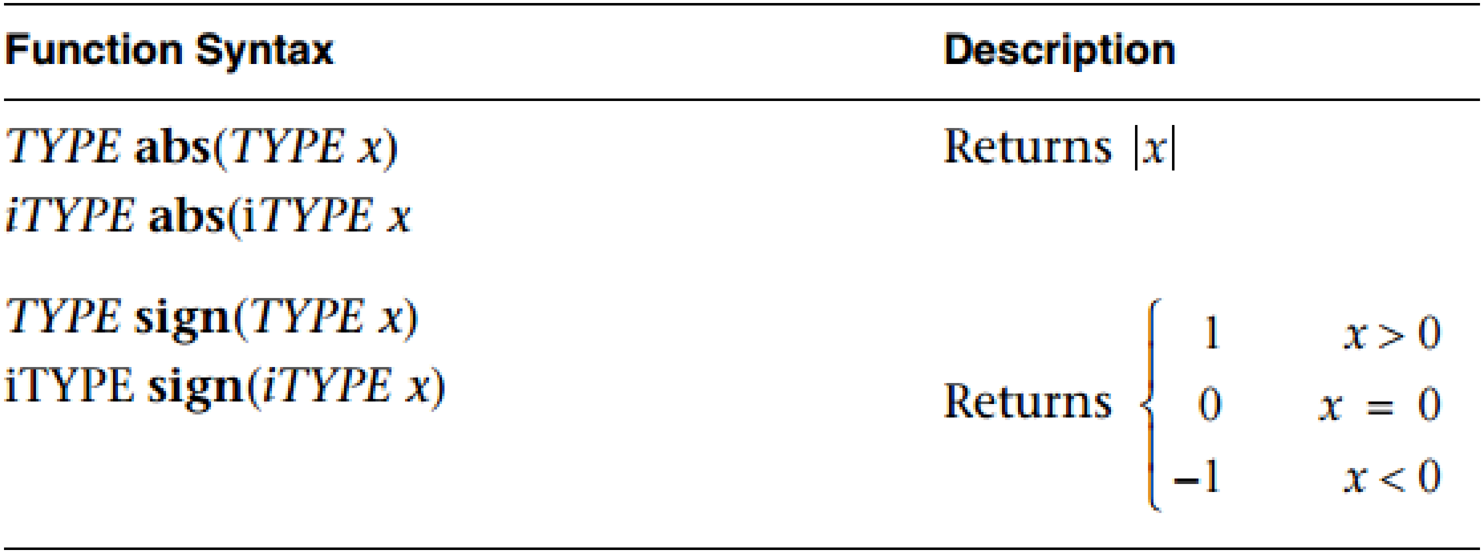
- Transcendental Functions



Function Syntax	Description
TYPE **pow**(*TYPE x*, *TYPE y*)	Returns xy. Results are undefined if $x < 0$, or if $x = 0$ and $y \leq 0$
TYPE **exp**(*TYPE x*)	Returns e^x.
TYPE **log**(*TYPE x*)	Returns $\ln(x)$. Results are undefined if $x \leq 0$
TYPE **exp2**(*TYPE x*)	Returns 2^x.
TYPE **log2**(*TYPE x*)	Returns $\log_2(x)$. Results are undefined if $x \leq 0$.
TYPE **sqrt**(*TYPE x*)	Returns $\sqrt{x}$. Results are undefined if $x \leq 0$.
TYPE **inversesqrt**(*TYPE x*)	Returns $\frac{1}{\sqrt{x}}$. Results are undefined if $x \leq 0$.

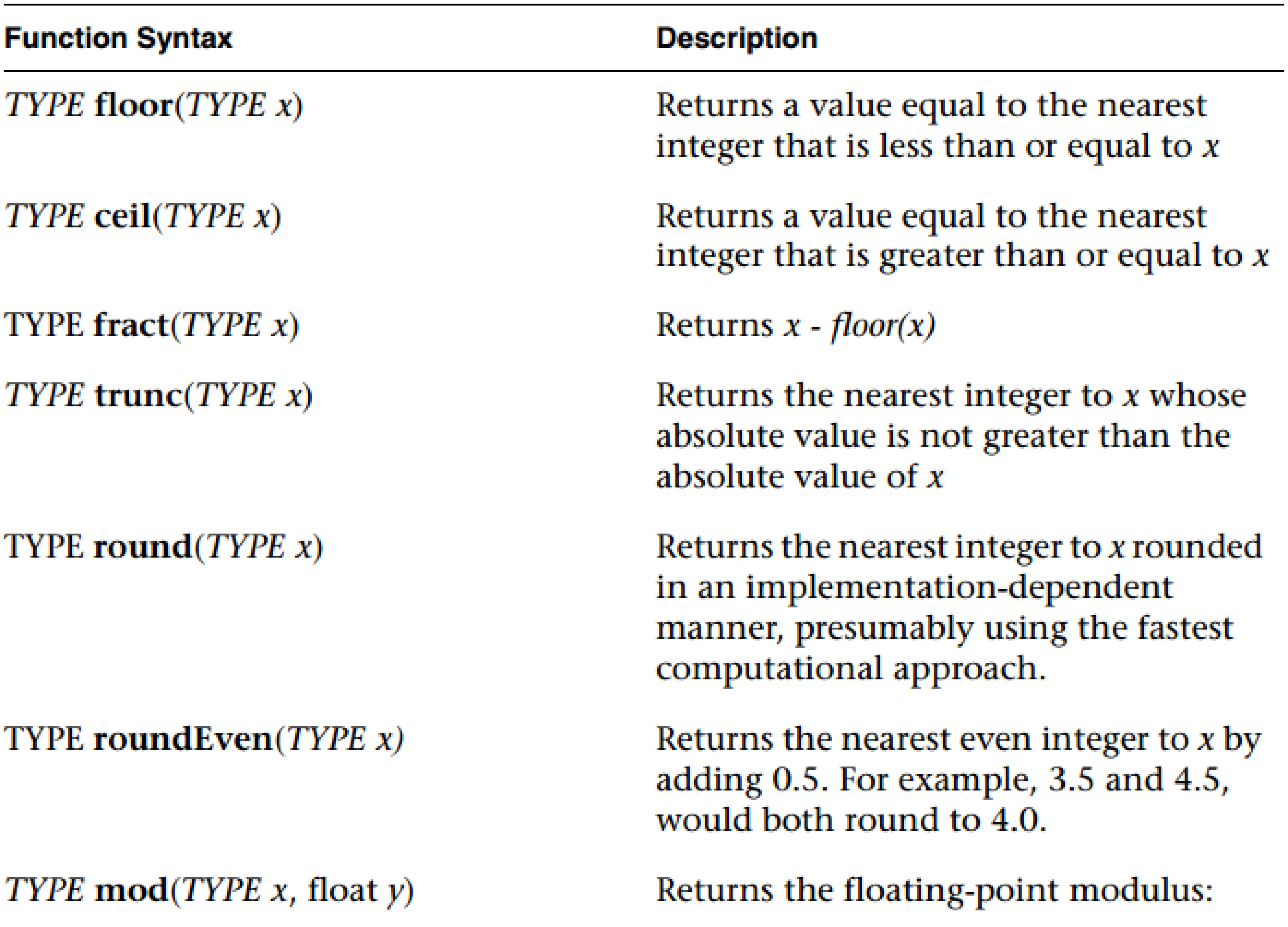
Built-In OpenGL Shading Language Functions

- Basic Numerical Functions



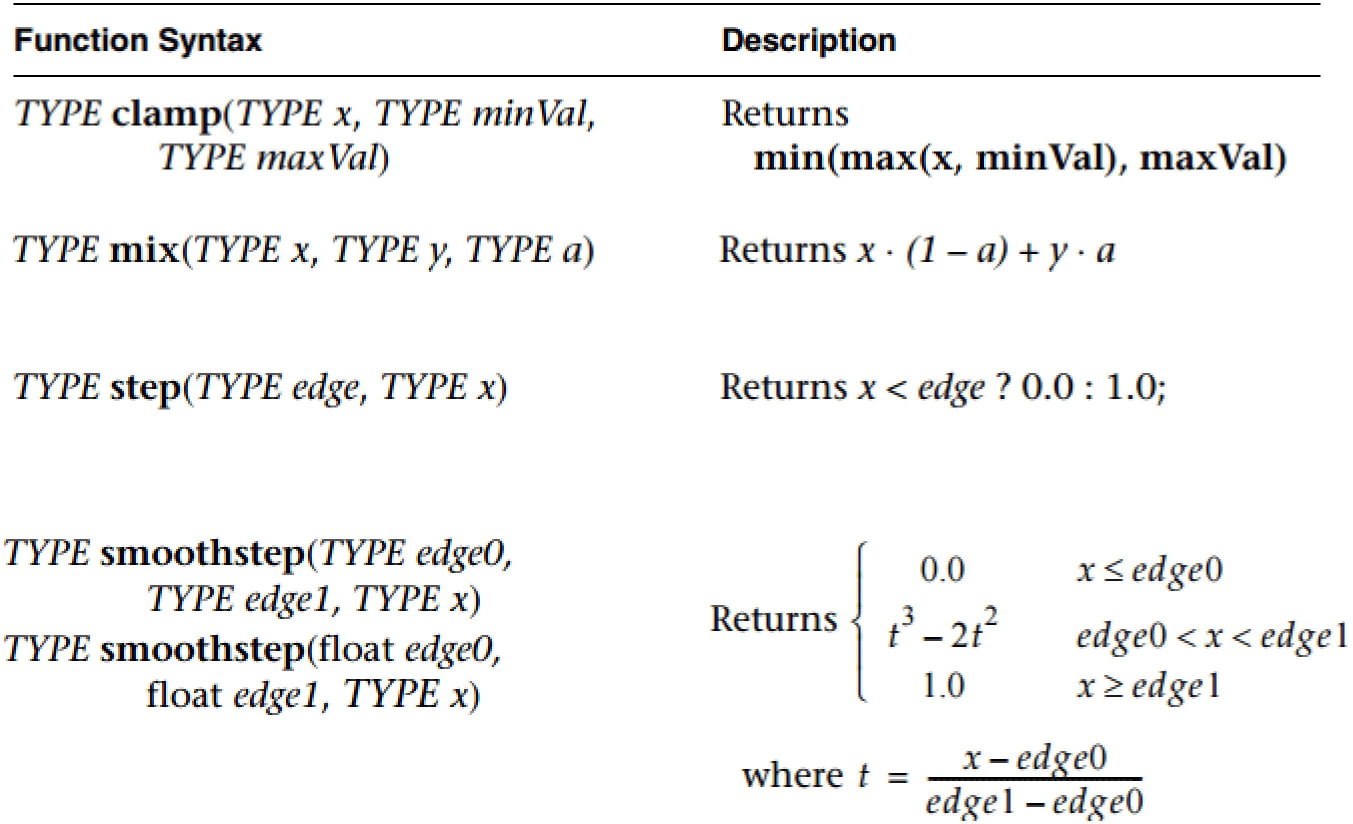
Function Syntax	Description
TYPE **abs**(*TYPE x*) *iTYPE* **abs**(i*TYPE x*	Returns $\lvert x \rvert$
TYPE **sign**(*TYPE x*) iTYPE **sign**(*iTYPE x*)	Returns $\begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$

Built-In OpenGL Shading Language Functions



Function Syntax	Description
TYPE **floor**(*TYPE x*)	Returns a value equal to the nearest integer that is less than or equal to *x*
TYPE **ceil**(*TYPE x*)	Returns a value equal to the nearest integer that is greater than or equal to *x*
TYPE **fract**(*TYPE x*)	Returns *x* - *floor(x)*
TYPE **trunc**(*TYPE x*)	Returns the nearest integer to *x* whose absolute value is not greater than the absolute value of *x*
TYPE **round**(*TYPE x*)	Returns the nearest integer to *x* rounded in an implementation-dependent manner, presumably using the fastest computational approach.
TYPE **roundEven**(*TYPE x*)	Returns the nearest even integer to *x* by adding 0.5. For example, 3.5 and 4.5, would both round to 4.0.
TYPE **mod**(*TYPE x*, float *y*)	Returns the floating-point modulus:

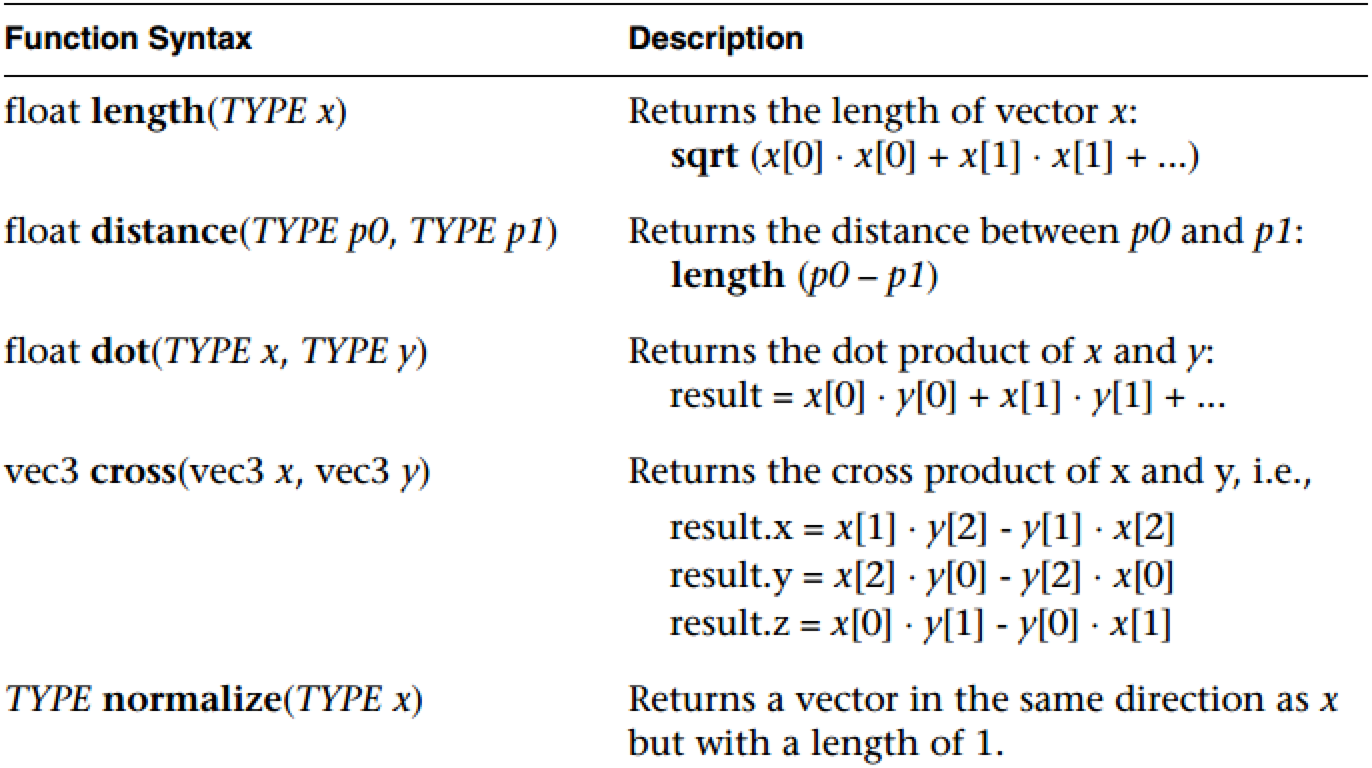
Built-In OpenGL Shading Language Functions



Function Syntax	Description
TYPE **clamp**(*TYPE x*, *TYPE minVal*, *TYPE maxVal*)	Returns **min(max(x, minVal), maxVal)**
TYPE **mix**(*TYPE x*, *TYPE y*, *TYPE a*)	Returns $x \cdot (1 - a) + y \cdot a$
TYPE **step**(*TYPE edge*, *TYPE x*)	Returns $x < edge$? 0.0 : 1.0;
TYPE **smoothstep**(*TYPE edge0*, *TYPE edge1*, *TYPE x*) *TYPE* **smoothstep**(float *edge0*, float *edge1*, *TYPE x*)	Returns $\begin{cases} 0.0 & x \le edge0 \\ t^3 - 2t^2 & edge0 < x < edge1 \\ 1.0 & x \ge edge1 \end{cases}$ where $t = \dfrac{x - edge0}{edge1 - edge0}$

Built-In OpenGL Shading Language Functions

- Vector Operations

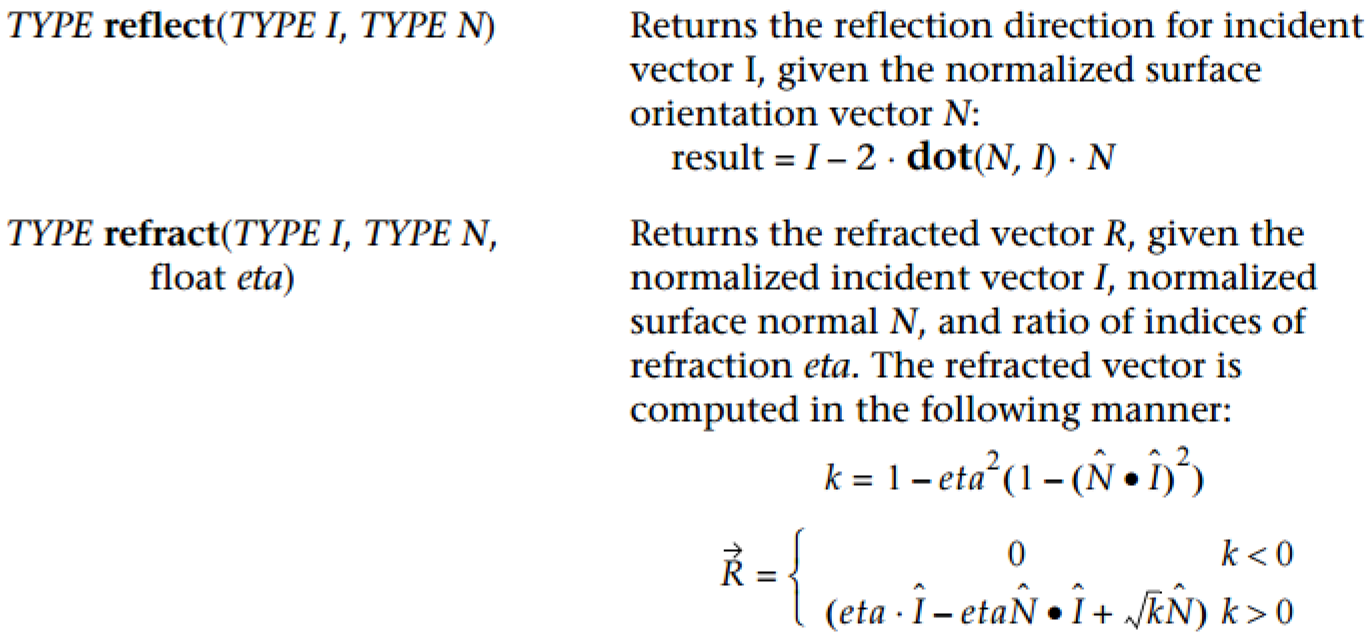


Function Syntax	Description
float **length**(*TYPE x*)	Returns the length of vector *x*: **sqrt** (*x*[0] · *x*[0] + *x*[1] · *x*[1] + ...)
float **distance**(*TYPE p0, TYPE p1*)	Returns the distance between *p0* and *p1*: **length** (*p0* – *p1*)
float **dot**(*TYPE x, TYPE y*)	Returns the dot product of *x* and *y*: result = *x*[0] · *y*[0] + *x*[1] · *y*[1] + ...

vec3 **cross**(vec3 *x*, vec3 *y*)	Returns the cross product of x and y, i.e.,
	result.x = *x*[1] · *y*[2] - *y*[1] · *x*[2]
	result.y = *x*[2] · *y*[0] - *y*[2] · *x*[0]
	result.z = *x*[0] · *y*[1] - *y*[0] · *x*[1]

TYPE **normalize**(*TYPE x*)	Returns a vector in the same direction as *x* but with a length of 1.

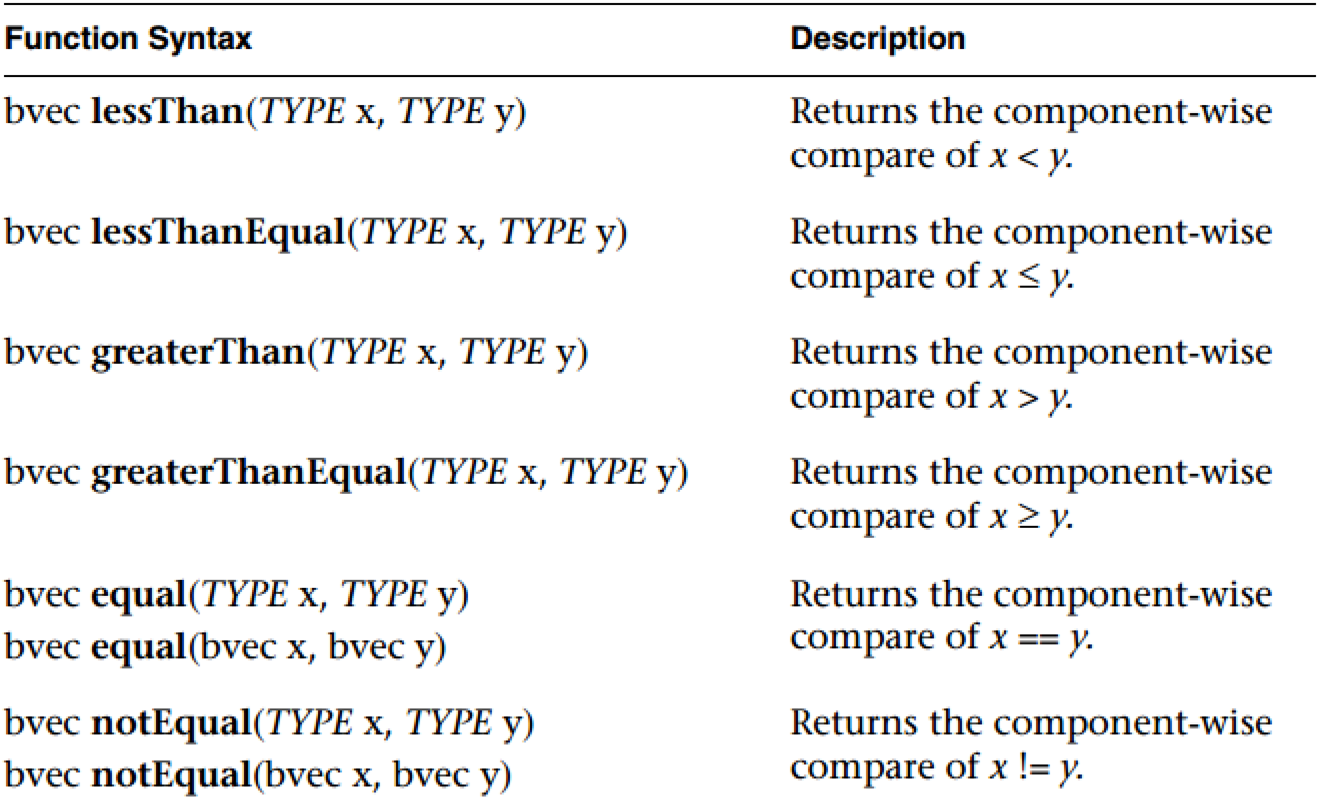
Built-In OpenGL Shading Language Functions



TYPE **reflect**(*TYPE I*, *TYPE N*)	Returns the reflection direction for incident vector I, given the normalized surface orientation vector *N*: result = $I - 2 \cdot \mathbf{dot}(N, I) \cdot N$
TYPE **refract**(*TYPE I*, *TYPE N*, float *eta*)	Returns the refracted vector *R*, given the normalized incident vector *I*, normalized surface normal *N*, and ratio of indices of refraction *eta*. The refracted vector is computed in the following manner: $k = 1 - eta^2(1 - (\hat{N} \bullet \hat{I})^2)$ $\vec{R} = \begin{cases} 0 & k < 0 \\ (eta \cdot \hat{I} - eta\hat{N} \bullet \hat{I} + \sqrt{k}\hat{N}) & k > 0 \end{cases}$

Built-In OpenGL Shading Language Functions

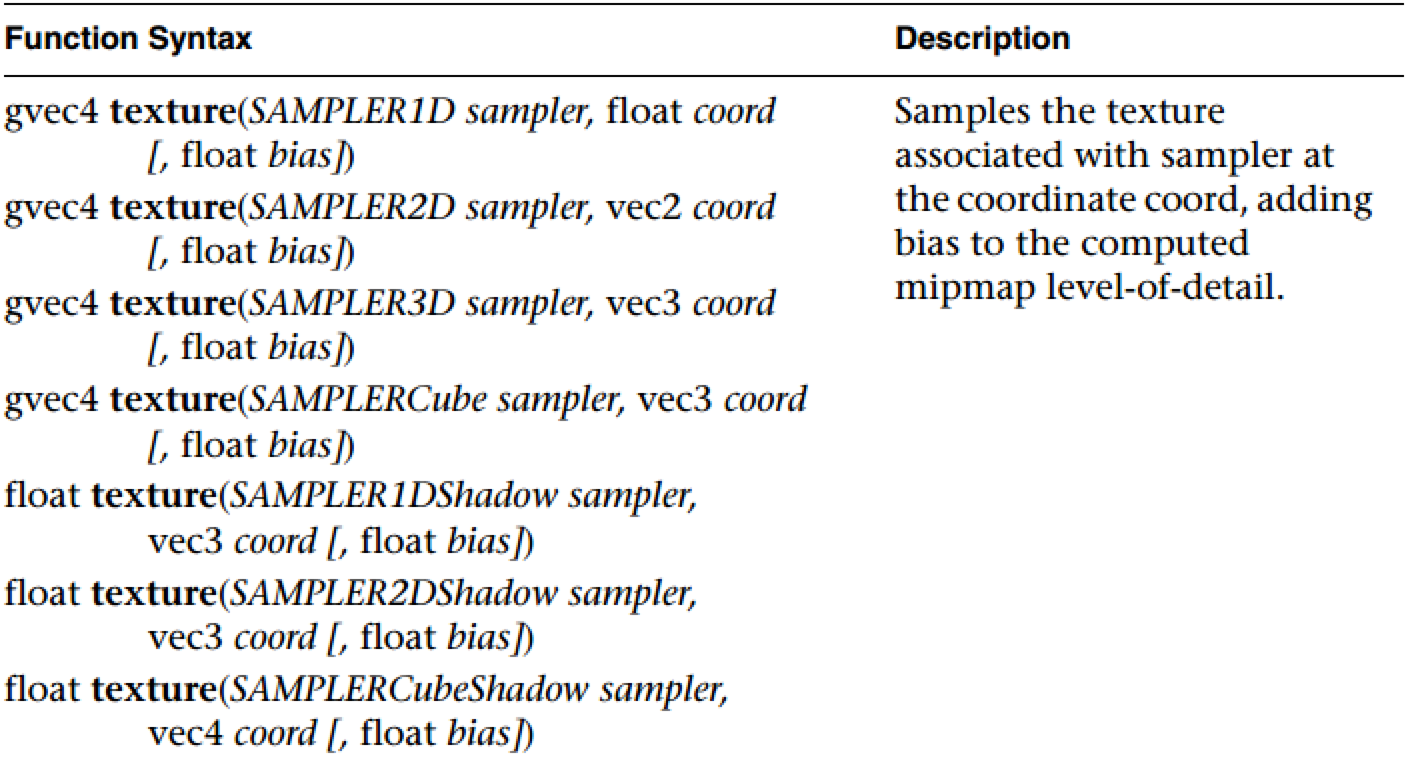
- Vector Component Relation Functions



Function Syntax	Description
bvec **lessThan**(*TYPE* x, *TYPE* y)	Returns the component-wise compare of $x < y$.
bvec **lessThanEqual**(*TYPE* x, *TYPE* y)	Returns the component-wise compare of $x \leq y$.
bvec **greaterThan**(*TYPE* x, *TYPE* y)	Returns the component-wise compare of $x > y$.
bvec **greaterThanEqual**(*TYPE* x, *TYPE* y)	Returns the component-wise compare of $x \geq y$.
bvec **equal**(*TYPE* x, *TYPE* y) bvec **equal**(bvec x, bvec y)	Returns the component-wise compare of $x == y$.
bvec **notEqual**(*TYPE* x, *TYPE* y) bvec **notEqual**(bvec x, bvec y)	Returns the component-wise compare of $x \mathrel{!=} y$.

Built-In OpenGL Shading Language Functions

- Basic Texture Access Function



Function Syntax	Description
gvec4 **texture**(*SAMPLER1D sampler*, float *coord* *[*, float *bias]*)	Samples the texture associated with sampler at the coordinate coord, adding bias to the computed mipmap level-of-detail.
gvec4 **texture**(*SAMPLER2D sampler*, vec2 *coord* *[*, float *bias]*)	
gvec4 **texture**(*SAMPLER3D sampler*, vec3 *coord* *[*, float *bias]*)	
gvec4 **texture**(*SAMPLERCube sampler*, vec3 *coord* *[*, float *bias]*)	
float **texture**(*SAMPLER1DShadow sampler*, vec3 *coord [*, float *bias]*)	
float **texture**(*SAMPLER2DShadow sampler*, vec3 *coord [*, float *bias]*)	
float **texture**(*SAMPLERCubeShadow sampler*, vec4 *coord [*, float *bias]*)	